

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2000-076086

(43)Date of publication of application : 14.03.2000

(51)Int.Cl.

G06F 9/46

(21)Application number : 10-244497

(71)Applicant : INTERNATL BUSINESS MACH CORP <IBM>

(22)Date of filing : 31.08.1998

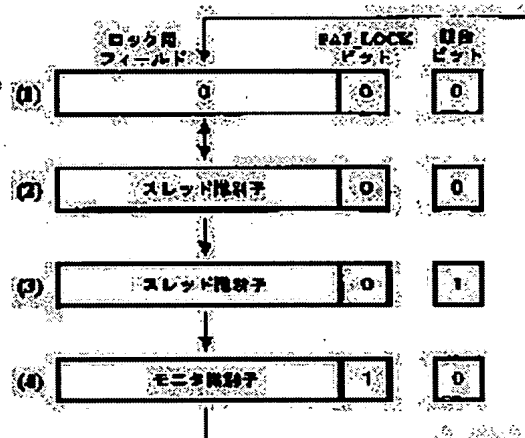
(72)Inventor : ONODERA TAMIYA
KOCHIYA KIYOKUNI

(54) METHOD AND DEVICE FOR LOCK MANAGEMENT OF OBJECT AND METHOD AND DEVICE FOR UNLOCKING OBJECT

(57)Abstract:

PROBLEM TO BE SOLVED: To provide a compound lock method not to decelerate the processing speed of a high-frequency path for locking, accessing and unlocking an object.

SOLUTION: In a state of making plural sleds existent, when managing lock to the object by storing a bit showing the kind of lock and the identifier of a sled, which acquires lock corresponding to the lock of a first kind, or the identifier of lock of a second kind in a storage area provided corresponding to the object, this method executes a step for judging the bit showing the kind of lock for a certain object held by the first sled shows the lock of the first kind or not when the second sled tries to acquire the lock of a certain object and a step for putting up a competition bit when the bit shows the lock of the first kind.



LEGAL STATUS

[Date of request for examination]

27.12.1999

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開2000-76086

(P2000-76086A)

(43) 公開日 平成12年3月14日 (2000.3.14)

(51) Int.Cl.⁷

G 0 6 F 9/46

識別記号

3 4 0

F I

G 0 6 F 9/46

テマコード* (参考)

3 4 0 H 5 B 0 9 8

審査請求 未請求 請求項の数18 O L (全 14 頁)

(21) 出願番号 特願平10-244497

(22) 出願日 平成10年8月31日 (1998.8.31)

(71) 出願人 390009531

インターナショナル・ビジネス・マシー
ズ・コーポレーション

INTERNATIONAL BUSIN
ESS MACHINES CORPO
RATION

アメリカ合衆国10504、ニューヨーク州
アーモンク (番地なし)

(74) 代理人 100086243

弁理士 坂口 博 (外1名)

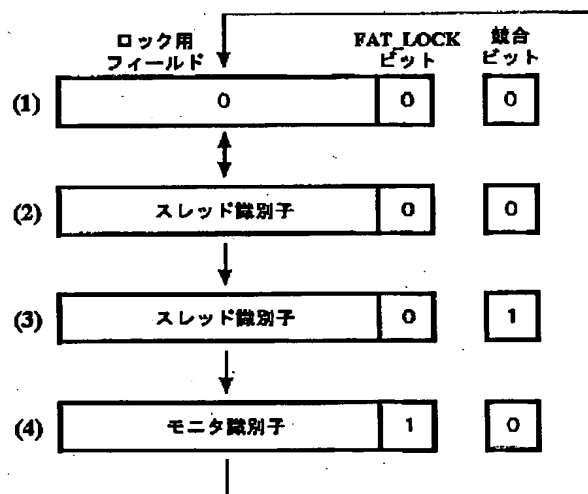
最終頁に続く

(54) 【発明の名称】 オブジェクトのロック管理方法及び装置、並びにオブジェクトのロック解除方法及び装置

(57) 【要約】 (修正有)

【課題】 オブジェクトにロックをし、アクセスし、アン
ロックするという高頻度パスの処理速度を低下させな
い、複合ロック方法。

【解決手段】 複数のスレッドが存在し得る状態におい
て、オブジェクトに対応して設けられた記憶領域にロッ
クの種類を示すビット及び第1の種類のロックに対応し
てロックを獲得したスレッドの識別子又は第2の種類の
ロックの識別子を記憶することによりオブジェクトへの
ロックを管理する場合に、第1のスレッドが保持してい
るあるオブジェクトへのロックを第2のスレッドが獲得
しようとした場合、あるオブジェクトのロックの種類を
示すビットが第1の種類のロックであることを示してい
るか判断するステップと、第1の種類のロックであるこ
とを示している場合には、競合ビットを立てるステッ
プとを実行する。



【特許請求の範囲】

【請求項1】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第1の種類のロックに対応してロックを獲得したスレッドの識別子又は第2の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する方法であって、

第1のスレッドが保持しているあるオブジェクトへのロックを第2のスレッドが獲得しようとした場合、前記あるオブジェクトの前記ロックの種類を示すビットが第1の種類のロックであることを示しているか判断するステップと、

前記第1の種類のロックであることを示している場合には、競合ビットを立てるステップと、
を含むロック管理方法。

【請求項2】前記第1のスレッドが前記あるオブジェクトへのロックを解除する際に、前記ロックの種類を示すビットが前記第1の種類のロックであることを示しているか判断するステップと、

前記あるオブジェクトのロックを保持しているスレッドが存在しないことを前記記憶領域に記憶するステップと、

前記ロックの種類を示すビットが前記第1の種類のロックであることを示している場合には、前記競合ビットが立っているか判断するステップと、

前記競合ビットが立っていないと判断された場合には、他の処理を実施せずにロック解除処理を終了するステップと、

をさらに含む請求項1記載のロック管理方法。

【請求項3】前記競合ビットが立っていると判断された場合には、オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構の排他制御状態に前記第1のスレッドが移行するステップと、待機しているスレッドへの通知操作を実行するステップと、
前記第1のスレッドが前記排他制御状態から脱出するステップと、

をさらに含む請求項2記載のロック管理方法。

【請求項4】前記第1の種類のロックとは、オブジェクトに対してロックを実施するスレッドの識別子を当該オブジェクトに対応して記憶することによりロック状態を管理するロック方式である、請求項1記載のロック管理方法。

【請求項5】前記第2の種類のロックとは、オブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である、請求項1記載のロック管理方法。

【請求項6】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロ

ックの種類を示すビット及び第1の種類のロックに対応してロックを獲得したスレッドの識別子又はオブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である第2の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する方法であって、

オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構の排他制御状態に第1のスレッドが移行するステップと、

あるオブジェクトの前記ロックの種類を示すビットが第1の種類のロックであることを示しているか判断するステップと、

前記あるオブジェクトの前記ロックの種類を示すビットが第1の種類のロックであることを示している場合、前記第1の種類のロックを第1のスレッドが獲得できるか判断するステップと、

前記第1の種類のロックを第1のスレッドが獲得できる場合には、前記あるオブジェクトに対応して設けられた記憶領域に第2の種類のロックを示すビット及び前記第2の種類のロックの識別子を記憶するステップと、
を含み、前記第1のスレッドは前記あるオブジェクトに対して必要な処理を終了した後に前記排他的状態を脱出することを特徴とするロック管理方法。

【請求項7】前記第1の種類のロックを第1のスレッドが獲得できない場合には、前記機構の待機状態に移行するステップとをさらに含む請求項6記載のロック管理方法。

【請求項8】前記あるオブジェクトのロックの前記ロックの種類を示すビットが第1の種類のロックであることを示していない場合、前記第1のスレッドは前記第2の種類のロックを獲得したとして前記排他的状態を脱出することなく処理を実施するステップと、
をさらに含む請求項6記載のロック管理方法。

【請求項9】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第1の種類のロックに対応してロックを獲得したスレッドの識別子又はオブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である第2の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する方法であって、

オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構の排他制御状態に第1のスレッドが移行するステップと、

あるオブジェクトの前記ロックの種類を示すビットが第2の種類のロックであることを示しているか判断する第1判断ステップと、

前記あるオブジェクトの前記ロックの種類を示すビット

3

が第2の種類のロックであることを示している場合、前記第1のスレッドは前記第2の種類のロックを獲得したとして前記排他的状態を脱出することなく処理を実施するステップと、
を含むロック管理方法。

【請求項10】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第1の種類のロックに対応してロックを獲得したスレッドの識別子又はオブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である第2の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する場合、前記ロックを解除する方法であって、
第1のスレッドが獲得しているあるオブジェクトのロックが第2の種類のロックであるか判断するステップと、
前記第1のスレッドが獲得しているあるオブジェクトのロックが第2の種類のロックである場合、オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構における待機状態のスレッドが存在しているか判断するステップと、
前記他のスレッドが存在していない場合、ロックを保持しているスレッドが存在しないことを前記記憶領域に記憶するステップとを含むロック解除方法。

【請求項11】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第1の種類のロックに対応してロックを獲得したスレッドの識別子又は第2の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する装置であって、
第1のスレッドが保持しているあるオブジェクトへのロックを第2のスレッドが獲得しようとした場合、前記あるオブジェクトの前記ロックの種類を示すビットが第1の種類のロックであることを示しているか判断する手段と、
前記第1の種類のロックであることを示している場合には、競合ビットを立てる手段と、
を含むロック管理装置。

【請求項12】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第1の種類のロックに対応してロックを獲得したスレッドの識別子又はオブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である第2の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する装置であって、
オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構の排他制御状態に第1のスレッドを移行させる手段と、

4

あるオブジェクトの前記ロックの種類を示すビットが第1の種類のロックであることを示しているか判断する手段と、

前記あるオブジェクトの前記ロックの種類を示すビットが第1の種類のロックであることを示している場合、前記第1の種類のロックを第1のスレッドが獲得できるか判断する手段と、

前記第1の種類のロックを第1のスレッドが獲得できる場合には、前記あるオブジェクトに対応して設けられた記憶領域に第2の種類のロックを示すビット及び前記第2の種類のロックの識別子を記憶する手段と、
を含み、前記第1のスレッドは前記あるオブジェクトに対して必要な処理を終了した後に前記排他的状態を脱出することを特徴とするロック管理装置。

【請求項13】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第1の種類のロックに対応してロックを獲得したスレッドの識別子又はオブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である第2の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する装置であって、
オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構の排他制御状態に第1のスレッドを移行させる手段と、
あるオブジェクトの前記ロックの種類を示すビットが第2の種類のロックであることを示しているか判断する手段と、

前記あるオブジェクトの前記ロックの種類を示すビットが第2の種類のロックであることを示している場合、前記第1のスレッドは前記第2の種類のロックを獲得したとして前記排他的状態を脱出することなく処理を実施する手段と、
を含むロック管理装置。

【請求項14】複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第1の種類のロックに対応してロックを獲得したスレッドの識別子又はオブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である第2の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する場合、前記ロックを解除する装置であって、
第1のスレッドが獲得しているあるオブジェクトのロックが第2の種類のロックであるか判断する手段と、
前記第1のスレッドが獲得しているあるオブジェクトのロックが第2の種類のロックである場合、オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構における待機状態のスレッドが

5

存在しているか判断する手段と、
前記他のスレッドが存在していない場合、所定の条件を
満たしているか判断する手段と、
前記所定の条件を満たしている場合に、ロックを保持し
ているスレッドが存在しないことを前記記憶領域に記憶
する手段とを含むロック解除装置。

【請求項15】複数のスレッドが存在し得る状態におい
て、オブジェクトに対応して設けられた記憶領域にロッ
クの種類を示すビット及び第1の種類のロックに対応し
てロックを獲得したスレッドの識別子又は第2の種類の
ロックの識別子を記憶することによりオブジェクトへの
ロックをコンピュータに管理させるプログラムを記憶す
る記憶媒体であって、

前記プログラムは前記コンピュータに、
第1のスレッドが保持しているあるオブジェクトへのロ
ックを第2のスレッドが獲得しようとした場合、前記あ
るオブジェクトの前記ロックの種類を示すビットが第1
の種類のロックであることを示しているか判断するステ
ップと、

前記第1の種類のロックであることを示している場合に
は、競合ビットを立てるステップと、
を実行させる、記憶媒体。

【請求項16】複数のスレッドが存在し得る状態におい
て、オブジェクトに対応して設けられた記憶領域にロッ
クの種類を示すビット及び第1の種類のロックに対応し
てロックを獲得したスレッドの識別子又はオブジェクト
へのアクセスを実施するスレッドをキューを用いて管理
するロック方式である第2の種類のロックの識別子を記
憶することによりオブジェクトへのロックをコンピュ
ータに管理させるプログラムを格納する記憶媒体であ
って、

前記プログラムは前記コンピュータに、
オブジェクトへのアクセスの排他制御と所定の条件が成
立した場合のスレッドの待機操作及び待機しているスレ
ッドへの通知操作とを可能にする機構の排他制御状態に
第1のスレッドが移行するステップと、
あるオブジェクトの前記ロックの種類を示すビットが第
1の種類のロックであることを示しているか判断するス
テップと、

前記あるオブジェクトの前記ロックの種類を示すビット
が第1の種類のロックであることを示している場合、前
記第1の種類のロックを第1のスレッドが獲得できるか
判断するステップと、

前記第1の種類のロックを第1のスレッドが獲得できる
場合には、前記あるオブジェクトに対応して設けられた
記憶領域に第2の種類のロックを示すビット及び前記第
2の種類のロックの識別子を記憶するステップと、
を実行させ、前記第1のスレッドは前記あるオブジェ
クトに対して必要な処理を終了した後に前記排他的状態
を脱出することを特徴とする、記憶媒体。

6

【請求項17】複数のスレッドが存在し得る状態におい
て、オブジェクトに対応して設けられた記憶領域にロッ
クの種類を示すビット及び第1の種類のロックに対応し
てロックを獲得したスレッドの識別子又はオブジェクト
へのアクセスを実施するスレッドをキューを用いて管理
するロック方式である第2の種類のロックの識別子を記
憶することによりオブジェクトへのロックをコンピュ
ータに管理させるプログラムを格納した記憶媒体であ
って、

10 前記プログラムは前記コンピュータに、
オブジェクトへのアクセスの排他制御と所定の条件が成
立した場合のスレッドの待機操作及び待機しているスレ
ッドへの通知操作とを可能にする機構の排他制御状態に
第1のスレッドが移行するステップと、

あるオブジェクトの前記ロックの種類を示すビットが第
2の種類のロックであることを示しているか判断する第
1判断ステップと、

前記あるオブジェクトの前記ロックの種類を示すビット
が第2の種類のロックであることを示している場合、前
記第1のスレッドは前記第2の種類のロックを獲得した
として前記排他的状態を脱出することなく処理を実施す
るステップと、を実行させる、記憶媒体。

【請求項18】複数のスレッドが存在し得る状態におい
て、オブジェクトに対応して設けられた記憶領域にロッ
クの種類を示すビット及び第1の種類のロックに対応し
てロックを獲得したスレッドの識別子又はオブジェクト
へのアクセスを実施するスレッドをキューを用いて管理
するロック方式である第2の種類のロックの識別子を記
憶することによりオブジェクトへのロックを管理する場合、
前記ロックをコンピュータに解除させるプログラム
を格納した記憶媒体であって、

前記プログラムは前記コンピュータに、
第1のスレッドが獲得しているあるオブジェクトのロッ
クが第2の種類のロックであるか判断するステップと、
前記第1のスレッドが獲得しているあるオブジェクトの
ロックが第2の種類のロックである場合、オブジェクト
へのアクセスの排他制御と所定の条件が成立した場合の
スレッドの待機操作及び待機しているスレッドへの通知
操作とを可能にする機構における待機状態のスレッドが
存在しているか判断するステップと、

前記他のスレッドが存在していない場合、ロックを保持
しているスレッドが存在しないことを前記記憶領域に記
憶するステップとを実行させる、記憶媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、複数のスレッドが
存在し得る状態における、オブジェクトへのロックに関
する。

【0002】

50 【従来技術】複数のスレッドが動作するプログラムで

オブジェクトへのアクセスを同期させるには、アクセスの前にオブジェクトをロック(lock)し、次にアクセスを行い、アクセスの後にアンロック(unlock)するようにプログラムのコードは構成される。このオブジェクトのロックの実装方法としては、スピンロック及びキューロックがよく知られている。また、最近ではそれらを組み合わせたもの(以下、複合ロックという)も提案されている。以下、それぞれについて簡単に説明する。

【0003】(1) スピンロック

スピンロックとは、オブジェクトに対してロックを実施 *10

```

10 /* ロック */
20 while (compare_and_swap(<o->lock,0,thread_id())==0)
30   yield();
40 /* oへのアクセス */
...
50 /* アンロック */
60 o->lock=0;
```

第20行及び第30行でロックを行っている。ロックが獲得できるまでyield()を行う。ここでyield()とは、現在のスレッドの実行を止め、スケジューラに制御を移すことである。通常、スケジューラは、他の実行可能なスレッドから1つを選び走らせるが、いずれまた、スケジューラは、もとのスレッドを走らせることになり、ロックの獲得に成功するまでwhile文の実行が繰り返される。yieldが存在していると、単にCPU資源の浪費だけでなく、実装がプラットフォームのスケジューリング方式に依存せざるを得ないため、期待どおりに動作するプログラムを書くことが困難になる。第20行におけるwhile文の条件であるcompare_and_swapは、オブジェクトoに用意されたフィールドo->lockの内容と、0とを比較して、その比較結果が真であればスレッドのID(thread_id())をそのフィールドに書き込むものである。よって、オブジェクトoに用意されたフィールドに0が格納されている場合には、ロックしているスレッドが存在しないことを表している。よって、第60行でアンロックする場合にはo->lockに0を格納する。なお、このフィールドは例えば1ワードであるが、スレッド識別子を格納するのに十分なビット数であればよい。

【0004】(2) キューロック

キューロックとは、オブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である。キューロックにおいては、スレッドTがオブジェクトoのロックに失敗した場合、Tは自分自身をoのキューに入れてサスペンドする。アンロックするコードには、キューが空か否かをチェックするコードが含まれ、空でなければキューから1つスレッドを取り出し、そのスレッドをリジュームする。このようなキューロックは、オペレーティング・システム(OS)のスケジューリング機構と一体になって実装され、OSのAPI(Application Programming Interface)として提供されて

*するスレッドの識別子を当該オブジェクトに対応して記憶することによりロック状態を管理するロック方式である。スピンロックでは、スレッドTがオブジェクトoのロック獲得に失敗した場合、すなわち他のスレッドSが既にオブジェクトoをロックしている場合、ロックに成功するまでロックを繰り返す。典型的には、compare_and_swapのようなアトミックなマシン命令を用いて、次のようにロック又はアンロックする。

【表1】

いる。例えば、セマフォやMutex変数などが代表的なものである。キューロックにおいては、スペースオーバーヘッドはもはや1ワードでは済まず、十数バイトとなるのが普通である。また、ロックやアンロックの関数の内部では、キューという共有資源が操作されるため、何らかのロックが獲得され又は解放されている点にも注意する必要がある。

【0005】(3) 複合ロック

マルチ・スレッド対応のプログラムは、マルチ・スレッドで実行されることを考慮して共有資源へのアクセスはロックにより保護するように書かれる。しかし、例えばマルチ・スレッド対応ライブラリがシングル・スレッドのプログラムから使用されるような場合もある。また、マルチ・スレッドで実行されてもロックの競合がほとんど発生しない場合もある。実際、Java(Sun Microsystems社の商標)のプログラムの実行履歴によると、多くのアプリケーションにおいて、オブジェクトへのアクセスの競合はほとんど発生していないという報告もある。

【0006】よって、「ロックされていないオブジェクトにロックをかけ、アクセスし、アンロックする」は高頻度に行われるパスであると考えられる。このパスは、スピンロックでは極めて効率よく実行されるが、キューロックでは時間的にも空間的にも効率が悪い。一方、高頻度ではないとはいえ、競合が実際に発生した場合、スピンロックではCPU資源が無益に消費されてしまうが、キューロックではそのようなことはない。

【0007】複合ロックの基本的なアイデアは、スピンロックのような処理が簡単なロック(軽量ロックと呼ぶ)とキューロックのような処理が複雑なロック(重量ロックと呼ぶ)をうまく組み合わせて、上記の高頻度パスを高速に実行しつつ、競合時の効率も維持しようというものである。具体的に言えば、最初に軽量ロックでの

ロックを試み、軽量ロックで競合した場合重量ロックに遷移し、それ以降は重量ロックを使用するものである。

【0008】この複合ロックでは、スピンロックと同様と、オブジェクトにはロック用のフィールドがあり、「スレッド識別子」又は「重量ロック識別子」の値、及び、いずれの値を格納しているかを示すブール値が格納される。

【0009】ロックの手順は以下のとおりである。

1) アトミックな命令 (例えば、`compare_and_swap`) で軽量ロック獲得を試みる。成功すればオブジェクトへのアクセスを実行する。失敗した場合、すでに重量ロックになっているか、又は軽量ロックのままだが他のスレッドがロックをかけているのかのいずれかであることが分かる。

2) 既に重量ロックになっていれば、重量ロックを獲得する。

```

10: void lock(o) {
20:     if (compare_and_swap(<o->lock, 0, thread_id()))
30:         return;
40:     while (! (o->lock < FAT_LOCK)) {
50:         yield();
60:         if (compare_and_swap (<o->lock, 0, thread_id())){
70:             inflate(o);
80:             return;
90:         }
100:    }
110:    fat_lock(o->lock)
120:    return;
130: }
150: void unlock (o){
160:     If (o->lock==thread_id())
170:         o->lock=0;
180:     else
190:         fat_unlock(o->lock);
200: }
220: void inflate(o){
230:     o->lock= alloc_fat_lock() | FAT_LOCK;
240:     fat_lock(o->lock);
250: }

```

【0012】表2に示された擬似コードは、第10行から第130行までがロック関数、第150行から第200行までがアンロック関数、第220行から第250行までがロック関数で用いられる `inflate` 関数を示している。ロック関数内では、第20行で軽量ロックが試みられる。もしロックが獲得されれば、当該オブジェクトへのアクセスを実行する。そして、アンロックする場合には、第160行でオブジェクトのロック用フィールドにスレッド識別子が入力されているので、第170行においてそのフィールドに0を入力する。このように高頻度パスはスピンロックと同じで高速に実行することができ

* 3) 軽量ロックで競合した場合、軽量ロックを獲得した上で重量ロックへ遷移し、これを獲得する (以下の説明では、`inflate` 関数において実行される。)

【0010】複合ロックには、3) における「軽量ロックの獲得」で `yield` するか否かで2種類の実装がある。これらを詳しく以下に説明する。なお、ロック用のフィールドは1ワードとし、さらに簡単のため「スレッド識別子」又は「重量ロック識別子」は常に0以外の偶数であるとし、ロック用のフィールドの最下位ビットが0ならば「スレッド識別子」、1ならば「重量ロック識別子」が格納される。

【0011】複合ロックの例1

軽量ロックの獲得において、`yield` する複合ロックの場合である。ロック関数は上の手順に従って以下のように書くことができる。

* 【表2】

る。一方、第20行でロックを獲得できない場合には、第40行で `while` 文の条件であるロック用フィールドの最下位ビットである `FAT_LOCK` ビットとロック用フィールドをビットごとにANDした結果が0であるか、すなわち `FAT_LOCK` ビットが0であるか (より詳しく言うと軽量ロックであるか) 判断される。もし、この条件が満たされていれば、第60行にて軽量ロックを獲得するまで `yield` する。軽量ロックを獲得した場合には、第220行以降の `inflate` 関数を実行する。`inflate` 関数では、ロック用フィールド `o->lock` に重量ロック識別子及び論理値1である `FAT_LOCK` ビ

ット入力する(第230行)。そして、重量ロックを獲得する(第240行)。もし、第40行で既にFAT_LOCKビットが1である場合には、直ぐに重量ロックを獲得する(第110行)。重量ロックのアンロックは第190行にて行われる。なお、重量ロックの獲得及び重量ロックのアンロックは、本発明とはあまり関係ないので説明を省略する。

【0013】この表2ではロック用フィールドの書き換えは常に軽量ロックを保持するスレッドにより実施される点に注意されたい。これは、アンロックでも同じである。yieldが発生するのは、軽量ロックでの競合時に限 *

```

10: void lock (o) {
20:     if (compare_and_swap (<o->lock, 0, thread_id()))
30:         return;
40:     monitor_enter (o);
50:     while (! (o->lock, < FAT_LOCK)){
60:         if (compare_and_swap(<o->lock, 0, thread_id())){
70:             inflate(o);
80:             monitor_exit(o);
90:             return;
100:         } else
110:             monitor_wait(o);
120:     }
130:     monitor_exit(o);
140:     fat_lock(o->lock);
150:     return;
160: }

180: void unlock (o) {
190:     if (o->lock == thread_id()) {
200:         o->lock=0;
210:         monitor_enter(o);
220:         monitor_notify(o);
230:         monitor_exit(o);
240:     } else
250:         fat_unlock(o->lock);
260: }

280: void inflate (o) {
290:     o->lock = alloc_fat_lock() | FAT_LOCK
300:     fat_lock(o->lock);
310:     monitor_notify_all(o);
320: }

```

【0015】モニタとは、Hoareによって考案された同期機構であり、オブジェクトへのアクセスの排他制御(enter及びexit)と所定の条件が成立した場合のスレッドの待機操作(wait)及び待機しているスレッドへの通知操作(notify及びnotify_all)とを可能にする機構である(Hoare, C.A.R. Monitors: An operating system structuring concept. Communications of ACM 17, 10 (Oct. 1974), 549-557 参照)。高だか1つのスレッドがモニタにエンタ(enter)することが許され

*定されている。

【0014】複合ロックの例2

軽量ロックの獲得において、yieldしない複合ロックの例を示す。軽量ロックが競合した場合にはウエイト(wait)する。軽量ロック解放時には、ウエイトしているスレッドに通知(notify)しなければならない。このウエイト及び通知のためには、条件変数やモニタあるいはセマフォを必要とする。以下の例ではモニタを使用して説明する。

【表3】

る。スレッドTがモニタmにエンタしようとした時、あるスレッドSが既にエンタしているならば、Tは少なくともSがmからイグジット(exit)するまで待たされる。このように排他制御がなされる。また、モニタmにエンタ中のスレッドTは、ある条件の成立を待つため、モニタmでウエイト(wait)することができる。具体的には、Tは陰にmよりイグジットしサスペンドする。陰にmよりイグジットすることにより、別のスレッドがモニタmにエンタすることができる点に注意されたい。一

方、モニタmにエンタ中のスレッドSは、ある条件を成立させた後に、モニタmに通知(notify)することができる。具体的には、モニタmでウエイト中のスレッドのうちのひとつUを起こす(wake up)する。それにより、Uはリジュームし、モニタmに陰にエンタしようとする。ここで、Sがmにエンタ中であるから、Uは少なくともSがmからイグジットするまで待たされる点に注意されたい。また、モニタmでウエイト中のスレッドが存在しない場合には、何も起こらない。notify_allは、ウエイト中のスレッドを全て起こす点を除いて、notifyと同じである。

【0016】表3において、第10行乃至第160行はロック関数、第180行乃至第260行はアンロック関数、第280行乃至320行はinflate関数を示している。ロック関数で複合ロックの例1と異なる点は、第40行でモニタにエンタする点、軽量ロックで競合した場合にyieldせずにウエイトする点(第110行)、重量ロックに遷移した際(第80行)及び重量ロックに遷移したことが確認された際(第130行)にはモニタからイグジットする点である。ここで、第130行ではモニタからイグジットし、第140行で重量ロックを獲得している点に注意されたい。

【0017】アンロック関数で複合ロックの例1と異なる点は、第210行乃至第230行においてモニタにエンタし、モニタで通知をし、モニタをイグジットする処理を実施している点である。これは、yieldをやめてモニタにおけるウエイトにしたためである。inflate関数では、notify_allが追加されている。これもyieldをやめてモニタにおけるウエイトにしたためである。なお、第290行は、alloc_fat_lock()で得られる重量ロック識別子と論理値1にセットされたFAT_LOCKビットをOR操作して、ロック用フィールドに入力する操作を示している。

【0018】表3を見れば、yieldは消滅しているが、アンロック時にウエイトしているスレッドがいるかもしれないので、通知(notify)という作業が入り、高頻度パスの性能が低下している。また、空間効率的には、モニタ又はモニタと同等な機能が余分に必要になっているが、重量ロックに遷移した後は不要になる。言いかえれば、モニタと重量ロックとは別に用意する必要がある。

【0019】複合ロックの例3

この例では、複合ロックの例1とは異なり、重量ロックとモニタとを別に用意せず、FAT_LOCKビットが重量ロックへの遷移を示しており且つモニタに入った場合には重量ロックを獲得したとして処理をする。例えば、David F.Bacon, Ravi Konuru, Chet Murthy, and Mauricio Serrano. Thin Locks: Featherweight Synchronization for Java. Proceedings of the SIGPLAN '98 Conference on Programming Language Design and Imp

lementation (1998), pp. 258-268を参照のこと。但し、この論文ではyieldが行われている。

【0020】

【発明が解決しようとする課題】本発明の目的は、高頻度パスの処理速度を低下させない、新規な複合ロック方法を提供することである。

【0021】また、yieldを用いずに、重量ロックとモニタとを別に用意することなく、FAT_LOCKビットが重量ロックへの遷移を示しており且つモニタに入った場合には重量ロックを獲得したとして処理できる、オブジェクトのロック方法を提供することも目的である。

【0022】さらに、上記の複合ロックでは、重量ロックから軽量ロックへの遷移は何等考慮されていない。よって、本発明では、重量ロックから軽量ロックへの遷移を可能にすることも目的である。

【0023】

【課題を解決するための手段】高頻度パスの処理速度を低下させないようにするため、本発明では少なくとも1ビットの競合ビットを導入する。これはロック用フィールドとは別に用意する。この競合ビットは、軽量ロックで競合が起きた場合にセットされ、軽量ロックから重量ロックへ遷移する際(inflate関数)にクリアされる。これをまとめると、複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第1の種類のロックに対応してロックを獲得したスレッドの識別子又は第2の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する場合に、第1のスレッドが保持しているあるオブジェクトへのロックを第2のスレッドが獲得しようとした場合、あるオブジェクトのロックの種類を示すビットが第1の種類のロックであることを示しているか判断するステップと、第1の種類のロックであることを示している場合には、競合ビットを立てるステップとを実行する。

【0024】このようにすると、第1のスレッドがあるオブジェクトへのロックを解除する際に、ロックの種類を示すビットが第1の種類のロックであることを示しているか判断するステップと、あるオブジェクトのロックを保持しているスレッドが存在しないことを記憶領域に記憶するステップと、ロックの種類を示すビットが第1の種類のロックであることを示している場合には、競合ビットが立っているか判断するステップと、競合ビットが立っていないと判断された場合には、他の処理を実施せずにロック解除処理を終了するステップとを実行できるようになる。すなわち、上で述べた高頻度パスの性能低下が競合ビットが立っているか否かのチェックだけに限定される。

【0025】一方、競合ビットが立っていると判断された場合には、オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機

15

しているスレッドへの通知操作とを可能にする機構の排他制御状態に第1のスレッドが移行するステップと、待機しているスレッドへの通知操作を実行するステップと、第1のスレッドが排他制御状態から脱出するステップとを実施する。

【0026】なお、第1の種類のロックとは、オブジェクトに対してロックを実施するスレッドの識別子を当該オブジェクトに対応して記憶することによりロック状態を管理するロック方式であり、第2の種類のロックとは、オブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である。

【0027】また、yieldを用いずに、重量ロックとモニタとを別に用意することなく、FAT_LOCKビットが重量ロックへの遷移を示しており且つモニタに入った場合には重量ロックを獲得したとして処理できるようにするため、複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第1の種類のロックに対応してロックを獲得したスレッドの識別子又はオブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である第2の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する場合には、オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構の排他制御状態に第1のスレッドが移行するステップと、あるオブジェクトのロックの種類を示すビットが第1の種類のロックであることを示しているか判断するステップと、あるオブジェクトのロックの種類を示すビットが第1の種類のロックであることを示している場合、第1の種類のロックを第1のスレッドが獲得できるか判断するステップと、第1の種類のロックを第1のスレッドが獲得できる場合には、あるオブジェクトに対応して設けられた記憶領域に第2の種類のロックを示すビット及び第2の種類のロックの識別子を記憶するステップとを実行し、第1のスレッドはあるオブジェクトに対して必要な処理を終了した後に排他的状態を脱出する。

【0028】第1の種類のロックを第1のスレッドが獲得できない場合には、モニタの待機状態に移行するステップをさらに実行する。あるオブジェクトのロックの種類を示すビットが第1の種類のロックであることを示していない場合、第1のスレッドは第2の種類のロックを獲得したとして排他的状態を脱出することなく処理を実施するステップとをさらに実行する。

【0029】また、上記目的のため、オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構の排他制御状態に第1のスレッドが移行するステップと、あるオブジェクトのロックの種類を示すビットが第2の種類のロックであることを示して

16

いるか判断するステップと、あるオブジェクトのロックの種類を示すビットが第2の種類のロックであることを示している場合、第1のスレッドは前記第2の種類のロックを獲得したとして排他的状態を脱出することなく処理を実施するステップとを実行する。既に、重量ロックに遷移している場合には、モニタにエンタすることにより重量ロックの獲得として処理を実施できる。

【0030】重量ロックから軽量ロックへの遷移を可能にするために、複数のスレッドが存在し得る状態において、オブジェクトに対応して設けられた記憶領域にロックの種類を示すビット及び第1の種類のロックに対応してロックを獲得したスレッドの識別子又はオブジェクトへのアクセスを実施するスレッドをキューを用いて管理するロック方式である第2の種類のロックの識別子を記憶することによりオブジェクトへのロックを管理する場合、ロックを解除するには、第1のスレッドが獲得しているあるオブジェクトのロックが第2の種類のロックであるか判断するステップと、第1のスレッドが獲得しているあるオブジェクトのロックが第2の種類のロックである場合、オブジェクトへのアクセスの排他制御と所定の条件が成立した場合のスレッドの待機操作及び待機しているスレッドへの通知操作とを可能にする機構における待機状態のスレッドが存在しているか判断するステップと、他のスレッドが存在していない場合、所定の条件を満たしているか判断するステップと、所定の条件を満たしている場合に、ロックを保持しているスレッドが存在しないことを記憶領域に記憶するステップとを実行する。

【0031】以上述べた処理のフローは、専用の装置として実施することも、また、コンピュータのプログラムとして実施することも可能である。さらに、このコンピュータのプログラムは、CD-ROMやフロッピー・ディスク、MO (Magneto-optic) ディスクなどの記憶媒体、又はハードディスクなどの記憶装置に記憶される。

【0032】

【発明の実施の形態】本発明の処理が実施されるコンピュータの例を図1に示す。コンピュータ1000は、CPU (1又は複数) 及びメインメモリを含むハードウェア100と、OS (Operating System) 200、アプリケーション・プログラム300を含む。OS 200は、アプリケーション・プログラム300として動作する複数のスレッドを可能にする能力を有する。また、OS 200はキューロックに必要な機能も提供する。また、アプリケーション・プログラム300は、モニタ機能、本発明のロック及びアンロック機能を含む。さらに、Java言語の場合には、Java VM (Virtual Machine) 310をOS 200上に設け、さらにその上でアプレット又はアプリケーション320を実行する場合もある。アプレット又はアプリケーション320もマルチ・スレッドで実行され得る。Java言語においては、J

17

ava VM310に、モニタ機能、本発明のロック及びアンロック機能が組み込まれる。なお、Java VM(310)はOS200の一部として組み込まれる場合もある。また、コンピュータ1000は補助記憶装置を有しない、いわゆるネットワークコンピュータ等でもよい。

【0033】本発明の処理を説明前に、高頻度パスの処理速度を低下させないための競合ビットを新たに導入する。図2に示したように、あるオブジェクトをロックしているスレッドが存在しない場合(1の場合)に、10は、ロック用フィールド及び競合ビット共に0が格納される。その後、あるスレッドがそのオブジェクトをロック(軽量ロック)すると、そのスレッドの識別子がロック用フィールドに格納される(2の場合)。もし、

18

このスレッド識別子のスレッドがロックを解放するまでに他のスレッドがロックを試みなければ(1に戻る)。ロックを解放するまでに他のスレッドがロックを試みると、軽量ロックにおける競合が発生したので、この競合を記録するため競合ビットを立てる(3の場合)。その後、重量ロックに移行した際には、競合ビットはクリアされる(4の場合)。可能であれば、(4)は(1)に移行する。なお、ロック用フィールドの最下位に軽量ロックと重量ロックのモードを表すビット(FAT_LOCKビット)設けるようにしたが、最上位に設けるようにしても良い。

【0034】上のような競合ビット及びロック用フィールドを用いた本発明の処理を以下に示す。

【表4】

```

10: void lock (Object* o){
20:     /* 軽量ロック */
30:     if (compare_and_swap (<o->lock, 0, thread_id()))
40:         return;
50:     /* 重量ロック及びモード遷移パス */
60:     MonitorId mon=obtain_monitor(o);
70:     monitor_enter(mon);
80:     /* モード遷移ループ */
90:     while (o->lock < FAT_LOCK ==0) {
100:         set_flg_bit(o);
110:         if (compare_and_swap (<o->lock, 0, thread_id()))
120:             inflate(o, mon);
130:         else
140:             monitor_wait (mon);
150:     }
160:
170: }
180:
190: void unlock (Object* o) {
200:     /* 軽量ロックパス */
210:     if ((o->lock < FAT_LOCK ==0)
220:         o->lock=0;
230:         if (test_flg_bit(o)) { /* 本発明のオーバーヘッド */
240:             MonitorId mon=obtain_monitor(o);
250:             monitor_enter(mon);
260:             if (test_flg_bit(o))
270:                 monitor_notify(mon);
280:             monitor_exit(mon);
290:         }
300:         return;
310:     }
320:     /* 重量ロックパス */
330:     x=o->lock
340:     if (xについてモニタでウェイトしているスレッドがない)
350:         if (所定の条件が満たされた)
360:             o->lock=0; /* 重量ロックから軽量ロックへの遷移 */

```

19

20

```

370:   monitor_exit( x < -FAT_LOCK );
380: }
390:
400:
410: void inflate (Object* o, MonitorId man) {
430:   monitor_notify_all (man);
440:   o->lock= (Word) man | FAT_LOCK;
450: }
460:
470:
480: MonitorId obtain_monitor(Object* o){
490:   Word word=o->lock;
500:   MonitorID man;
510:   if (word < FAT_LOCK)
520:     man = word < -FAT_LOCK;
530:   else
540:     man = lookup_monitor(o);
550:   return man;
560: }

```

【0035】本発明で導入された競合ビットは表4では `flc_bit` として示されている。では、表4の内容を詳細に説明する。表4は大きく分けて4つの部分からなる。ロック関数の部分（第10行乃至第170行）、アンロック関数の部分（第190行乃至第380行）、軽量ロックから重量ロックへの遷移である `inflate` 関数の部分（第410行乃至第450行）、及びモニタの識別子を取得する `obtain_monitor` 関数の部分（第480行乃至第560行）である。

【0036】(1) ロック関数

第10行から始まったオブジェクト `o` に対するロック関数の処理では、まず軽量ロックの取得を試みる（第30行）。この軽量ロックの取得には、例えば `compare_and_swap` のようなアトミックな命令を用いる。この命令では、第1の引き数と第2の引き数が同じ値の場合、第3の引き数を格納するものである。ここでは、オブジェクト `o` のロック用フィールドである `o->lock` が0に等しい場合には、`thread_id()` によりスレッド識別子を取得して、ロック用フィールド `o->lock` に格納する。図2の(1)から(2)への遷移を実施したのである。そして、必要な処理を実施するため、リターンする（第40行）。もし、オブジェクト `o` のロック用フィールドである `o->lock` が0に等しくない場合には、軽量ロックの取得は失敗し、第60行に移行する。ここまでの処理は従来技術と同じである。

【0037】次に、モニタ識別子を取得する `obtain_monitor(o)` 関数の値を `man` という変数に代入し（第60行）、スレッドはそのモニタの排他制御状態に移行しようとする。すなわちモニタ (`monitor`) にエンタ (`enter`) しようとする（第70行）。もし、排他制御状態に移行することができれば、以下の処理を実施し、もしで

きなかった場合には、できるまでこの段階で待つ。次に、`while` 文の条件を判断する。すなわち、ロック用フィールド `o->lock` と `FAT_LOCK` ビットのビットごとのANDを実施し、`FAT_LOCK` ビットが立っているか判断する（第90行）。ここでは、現在重量ロックに移行しているのか、軽量ロック中なのかを判断している。もし、`FAT_LOCK` ビットが立っていなければ（軽量ロック中）、この計算の結果は0となるから、`while` 文以下の処理を実施する。一方、`FAT_LOCK` ビットが立っている場合（重量ロック中）、`while` 文以下の処理を実施せずに、モニタにエンタした状態のままになる。このように `FAT_LOCK` ビットが立っている場合に、モニタにエンタできた場合には、本発明では重量ロックを取得できたということを意味しており、このモニタからイグジット (`exit`) することなく（すなわち排他制御状態を脱出することなく）、このスレッドはオブジェクトに対する処理を実施する。

【0038】では、第90行で `FAT_LOCK` ビットが立っていないと判断された場合には、軽量ロックの競合が発生していることを意味するので、`flc_bit` をセットする（第100行、`set_flg_bit(o)`）。ここで、図2の(2)から(3)への遷移を実施したのである。そして、もう一度軽量ロックを取得できるか判断する（第110行）。もし、軽量ロックを取得できる場合には軽量ロックから重量ロックへの遷移のための `inflate` 関数の処理を実施する（第120行）。一方、軽量ロックが取得できない場合には、モニタの待機状態 (`wait`) に移行する（第140行）。モニタの待機状態は、先にモニタの説明の部分で述べたが、モニタから脱出してサスペンドするものである。このように、軽量ロックで競合が生じると、競合ビットである `flc_bit` が

セットされ、軽量ロックを取得できない場合には、モニタの待機状態に移行する。この待機状態に入ると、後に `inflate` 関数の処理又はアンロックする際に通知 (`notify` 又は `notify_all`) を受けることになる。

【0039】(2) `inflate` 関数

では、第410行乃至第450行の `inflate` 関数の処理を説明する。ここではまず、競合ビットがクリアされる (第420行、`clear_flg_bit`)。そして、モニタの通知操作 (`monitor_notify_all`) を実施する (第430行)。ここでは、待機状態の全てのスレッドに起きる (`wake up`) よう通知する。そして、ロック用フィールド `o->lock` に、モニタの識別子を格納した変数 `mon` とセットされた `FAT_LOCK` ビットをビットごとに OR した結果を格納する (第440行、`mon | FAT_LOCK`)。すなわち、図2の(3)から(4)の状態に移移させたのである。これで軽量ロックから重量ロックへの移移は完了する。なお、第120行の処理が終了すると、再度 `while` 文の条件をチェックすることになるが、既に `FAT_LOCK` ビットが立っているの、この場合には `while` 文から脱出して、モニタにエンタしたままとなる。すなわち、`while` 文の中の処理を実行しない。

【0040】通知を受けた全てのスレッドは第140行において陰にモニタにエンタしようとするが、モニタにエンタする前に待機することになる。これは、通知を行ったスレッドはアンロック処理を実施するまでモニタからイグジットしていないからである。

【0041】(3) アンロック関数

では、次に第190行乃至第380行のアンロック関数の処理について説明する。このアンロック関数は軽量ロックのアンロックと、重量ロックのアンロックを取扱う。重量ロックにおけるアンロックは、図2の(4)から(1)への移移を取扱うものである。

【0042】(3-1) 軽量ロックのアンロック

軽量ロックのアンロックでは、まず、ロック用フィールド `o->lock` と `FAT_LOCK` ビットのビットごとの AND を計算し、その値が0であるか判断する (第210行)。これは、第90行の `while` 文の条件と同じであって、軽量ロック中であるかどうか判断するものである。もし、軽量ロック中である場合には、`o->lock` に0を格納する (第220行)。これにより、ロックを保持しているスレッドが存在しないことが記録される。そして、競合ビットが立っているか判断する (第230行、`test_flg_bit`)。もし、軽量ロックで競合が生じていなくとも、第230行のみは実施しなければならない。よって、本発明における高頻度パスの唯一のオーバーヘッドがこの第230行である。競合ビットが立っていない場合には、他の処理を実施せずにアンロック処理を終了する (第300行)。

【0043】もし、競合ビットが立っている場合には、第60行及び第70行と同じように、変数 `mon` にモニ

タの識別子を格納し (第240行)、当該モニタ識別子のモニタにエンタしようとする (第250行)。すなわち、そのスレッドはモニタの排他制御状態に入ろうとする。もしモニタにエンタできた場合には、もう一度、競合ビットが立っていることを確認し (第260行)、もし立っていれば、モニタにおいて待機状態のスレッドの1つに起動を通知する (第270行、`monitor_notify(mon)`)。なお、モニタにエンタできない場合には、モニタにエンタできるまで待機する。そして通知を行ったスレッドは、モニタの排他制御状態から脱出する (第280行、`monitor_exit(mon)`)。

【0044】第270行で通知を受けたスレッドは、第140行で陰にモニタにエンタする。そして第80行に戻りその処理を実施する。通常、第270行で通知を受けたスレッドは、通知を行ったスレッドがモニタの排他制御状態を脱出した後にモニタの排他制御状態に入り、競合ビットを立てた後に、軽量ロックを取得し、`inflate` 関数の処理を実施することにより重量ロックに移移する。

【0045】(3-2) 重量ロックのアンロック

もし、第210行で `FAT_LOCK` ビットが立っていることが分かった場合には、第330行に処理は移行する。第330行では、ロック用フィールドの内容を変数 `x` に格納する。そして、モニタにおける待機状態 (`wait`) にあるスレッドが他に存在しないかを判断する (第340行)。もし、存在しない場合には、所定の条件を満たしているか判断する (第350行)。所定の条件には、重量ロックから脱出しない方がよいような条件があればそのような条件を設定する。但し、本ステップは実行しなくてもよい。もし、所定の条件を満たしている場合には、ロック用フィールド `o->lock` を0にする (第360行)。すなわち、ロックを保持しているスレッドが存在しないことをロック用フィールドに格納する。そして、変数 `x` の `FAT_LOCK` ビット以外の部分に格納されたモニタ識別子のモニタからイグジットする (第370行)。`x < -FAT_LOCK` は、`FAT_LOCK` ビットを反転させたものと `x` とのビットごとの AND である。これにより、モニタにエンタしようとして待機していたスレッドが、モニタにエンタできるようになる。

【0046】(4) モニタ識別子を取得する `obtain_mon` 関数

この関数では、まず、`word` という変数にロック用フィールドの内容を格納する (第490行)。そして、モニタの識別子を格納する変数 `mon` を用意し (第500行)、`FAT_LOCK` ビットが立っているか判断する (第510行、`word < FAT_LOCK`)。もし、`FAT_LOCK` ビットが立っているようであれば、変数 `mon` に `word` の `FAT_LOCK` ビット以外の部分を格納する (第520行、`word < -FAT_LOCK`)。一方、`FAT_LOCK` ビットが立っていない場合には、関数 `look_u`

p_monitor(o)を実行する(第530行)。この関数は表4で説明は省略しているが、オブジェクトとモニタの関係を記録したハッシュ・テーブルを有していることを前提とし、基本的にはこのテーブルをオブジェクトoについて検索して、モニタの識別子を取得する。もし、必要があれば、モニタを生成し、そのモニタの識別子をハッシュ・テーブルに格納した後にモニタ識別子を返す。いずれにしても、変数monに格納されたモニタの識別子を返す。

【0047】従来技術の欄で説明した従来技術と表4を比較すると、競合ビットを導入した他に、第150行乃至第170行の間に何等の処理が存在していない点、及び第320行乃至第370行の重量ロックから軽量ロックへの遷移が存在している点、が大きく異なる。競合ビットを導入したことにより第230行のチェックが必要になったが、競合ビットを導入しなければ、従来技術のような、より大きなペナルティを受ける。また、FAT_LOCKビットが立っており且つモニタの排他制御状態に移行することができた場合には重量ロックを獲得しているということにしたため、モニタの他に重量ロックの機構を用意する必要がなくなり、且つモニタの排他的状態からの脱出及び重量ロックの獲得といった処理をなくし、それにより処理を高速化することもできるようになった。また、重量ロックから軽量ロックへの遷移(図2の(4)から(1))を設けたことにより、低負荷な高頻度パス(図2の(1)と(2)の間の遷移)を実行できるような状態に戻ることができた。

【0048】以下に、競合ビットを表4内の第100行でセットし、第230行でチェックすることで何等の問題が生じないということについて述べておく。最初に、「競合ビットは、inflate関数でのみクリアされる」ということを確認しておく。

【0049】そして、スレッドTがウエイト(wait)したとする。スレッドTが必ず通知(notify)を受けることを、次の2つの場合に分けて説明する。

(1)その後inflate関数が実行される場合。
inflate関数が実行されると、第430行目でnotify_allが実行される。すなわち、Tはnotifyを受ける。

(2)inflate関数が実行されない場合。
Tがウエイト(wait)したのは、第110行目における軽量ロック獲得に失敗したからである。第110行目の失敗の時点を考えて、この時点で別なスレッドSが軽量ロックを保持している、すなわち、Sはアンロック関数の第220行目の実行に到達していない。また、Tがウエイト(wait)前にセットした競合ビットは、inflate関数が実行されない場合を考えているので、上で確認

した事項により、セットされたままである。Sはいずれアンロック関数の第220行目に到達し、次の競合ビットのチェックを実行するが、このチェックは必ず成功する。すなわち、TはSにより通知(notify)される。

【0050】また、図2における(4)から(1)の遷移を導入した。これは、1)軽量ロックを獲得するためには第30行のcompare_and_swapを成功しなければならないが、他のスレッドが重量ロックを獲得している限り、第30行のcompare_and_swapは成功しないので、重量ロックを他のスレッドが獲得している時には軽量ロックを獲得することは不可能であることが保証され、2)重量ロックを獲得するためには、モニタにエンタしてwhile文の条件が満たされない必要があるが、他のスレッドが軽量ロックを保持している限り、必ずwhile文の条件が満たされてしまうので、軽量ロックを他のスレッドが獲得している時にはモニタにエンタできても重量ロックを獲得することは不可能であることが保証されるので、安全な処理である。

【0051】

【効果】高頻度パスの処理速度を低下させない、新規な複合ロック方法を提供することができた。

【0052】また、yieldを用いずに、重量ロックとモニタとを別に用意することなく、FAT_LOCKビットが重量ロックへの遷移を示しており且つモニタに入った場合には重量ロックを獲得したとして処理できる、オブジェクトのロック方法を提供することもできた。

【0053】さらに、上記の複合ロックでは、重量ロックから軽量ロックへの遷移は何等考慮されていない。よって、本発明では、重量ロックから軽量ロックへの遷移を可能にすることもできた。

【図面の簡単な説明】

【図1】本発明の処理が実施されるコンピュータの一例を示す図である。

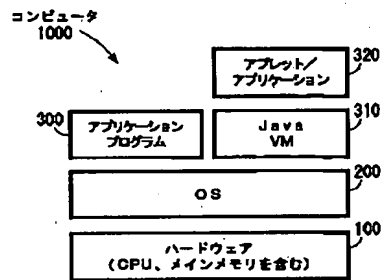
【図2】モードの遷移、並びに各モードにおけるロック用フィールド(FAT_LOCKビットを含む)及び競合ビットの状態を説明するための図である。なお、

(1)はロックなし、(2)は軽量ロックで競合なし、(3)は軽量ロックで競合あり、(4)は重量ロックの状態を示す。

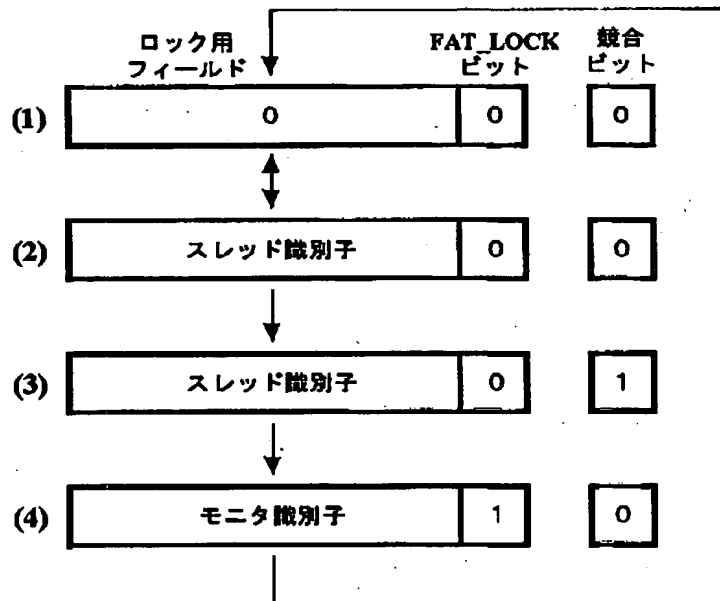
【符号の説明】

1000 コンピュータ
100 ハードウェア
200 OS
300 アプリケーション・プログラム
310 Java VM
320 Java アプレット/アプリケーション

【図1】



【図2】



フロントページの続き

(72)発明者 小野寺 民也
 神奈川県大和市下鶴間1623番地14 日本ア
 イ・ビー・エム株式会社 東京基礎研究所
 内

(72)発明者 河内谷 清久仁
 神奈川県大和市下鶴間1623番地14 日本ア
 イ・ビー・エム株式会社 東京基礎研究所
 内

Fターム(参考) 5B098 GA05 GD16

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

☐ BLACK BORDERS

☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES

☒ FADED TEXT OR DRAWING

☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING

☐ SKEWED/SLANTED IMAGES

☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS

☐ GRAY SCALE DOCUMENTS

☐ LINES OR MARKS ON ORIGINAL DOCUMENT

☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY

☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.